NS32CG16 Graphics Applications Note 3 — Emulation Software for Series 32000

Series 32000

NS32CG16 Graphics Applications Note 3

NS32CG16 Emulation Software for Series 32000

1. Introduction

The NS32CG16, a high speed graphics-specific version of the *Series 32000* family has a number of instructions which make graphics processing much easier. Since these instructions are available only in the NS32CG16, this emulation software was developed to permit easy migration of graphics software between all members of the *Series 32000* family.

For additional information on *Series 32000* programming, please see the *Series 32000 Programmer's Reference Manual*. For additional information on
these emulation routines and conventions, please see the NS32CG16 *Programmer's Reference Manual Supplement*.

2. Description

The emulation routines have been separated into three libraries: a pure emulation library, containing only *Series 32000* instructions; a pure native mode NS32CG16 library, containing the native NS32CG16 instructions; and an
automatic library, which uses the native mode NS32CG16 instructions if running on a NS32CG16, or emulation routines if running on another *Series 32000* processor. The automatic library uses a small routine called at run time to determine the processor type, and sets a memory variable so the emulation routines can quickly determine the processor type.

Each of the functions, where applicable, is furnished with a C front end, permitting the user to call the emulation routines directly from the C language. Since the *TBITS* instruction returns substantial information in the PSR, this routine is not represented by a C front end. Low-level assembly language entry points are also available for all instruction emulations.

The user is cautioned that these are *functional equivalents* for the NS32CG16 instructions. While each emulation produces the same effect as the
equivalent NS32CG16 instruction, the return register and/or flag contents may not be the same. This is due to the fact that each routine has been carefully optimized for speed.

3. Implementation

Each emulation routine is assembled with each of three flags, by running the source assembly language through the C preprocessor. This is done in the Makefile (see attached listing). The NOSOFT flag assembles the file using no software emulation, only NS32CG16 graphics instructions. The EMULATE flag assembles the file using ONLY software emulation, and generating no NS32CG16-specific instructions. The AUTO flag assembles the
file using BOTH software emulation and native mode instructions.
Each of the source files (see attached listings) is assembled, then made into a linkable archive with the ar command. When all archives have been created, they are copied to the /lib directory. Once the files have been moved to the /lib directory, they are accessible with the -l switch on the C compiler, and the ln linker, as in the following example:

```
$ cc testfile.c -o testemul -lcgemul    # Emulation library
$ cc testfile.c -o testnat  -lcgnat     # Native mode library
$ cc testfile.c -o testauto -lcgauto    # Automatic library
```

The first example uses the software emulation library, and thus will run on any *Series 32000* processor. The second example uses the NS32CG16 native mode library, and will run correctly only on the NS32CG16. The third example uses the automatic library, and will run on any *Series 32000* processor.

Each of the libraries has a specific purpose. The native mode library runs

the fastest on the NS32CG16, but does not work with other *Series 32000* processors. The automatic library should be used if the target processor of the code is not known. If on an NS32CG16, the emulation will run fast, since it will use the native mode instructions. It will not run as fast as the native mode library, however, due to the compare-and-branch over the native mode instructions. The emulation library will run on any *Series 32000* processor, but will not take advantage of NS32CG16-specific instructions. If the target is NOT an NS32CG16, this library will run slightly faster than the automatic library, again due to the compare-and-branch. If the Makefile is modified to include a "-DNS32532", then optimization will occur for the NS32532.

The instructions represented by this emulation library are as follows:

NS32CG16 instruction Emulation name Description
bband bband BitBlt AND function
bband -s bbandcs BBAND, complement source
bband da bbandda BBAND, decrement address
bband da,-s bbanddacs BBAND, compl. src, decr. addr.
bbfor bbfor BitBlt Fast OR function
bbor bbor BitBlt OR function
bbor -s bborcs BBOR, complement source
bbor da bborda BBOR, decrement address
bbor da,-s bbordacs BBOR, compl. src, decr. addr.
bbstod bbstod BitBlt Source TO Destination
bbstod -s bbstodcs BBSTOD, complement source
bbstod da bbstodda BBSTOD, decrement address
bbstod da,-s bbstoddacs BBSTOD, compl. src, decr. addr.
bbxor bbxor BitBlt XOR function
bbxor-s bbxorcs BBXOR, complement source
bbxorda bbxorda BBXOR, decrement address
bbxorda,-s bbxordacs BBXOR, compl. src, decr. addr.
bitwt bitwt Bit aligned Word Transfer (OR)
extblt (no pre-read) extblt External BitBlt (see listing)
extblt (pre-read) extbltp External BitBlt (see listing)
movmpb movmpb Move Multiple Pattern (byte)
movmpw movmpw Move Multiple Pattern (word)
movmpd movmpd Move Multiple Pattern (double word)
NS32CG16 instruction Emulation name Description
sbitps sbitps Set Bit Perpendicular String
sbits sbits Set Bit String
tbits 0 tbits0 (no C call) Test Bit String (zero)
tbits 1 tbits1 (no C call) Test Bit String (one)

- testcg Test for NS32CG16

The testcg function permits the user to test for the presence of the NS32CG16 (returns true if present, zero otherwise). It also initializes the memory variable used in the automatic emulation library. This function MUST
be called prior to the first call to the library if the automatic emulation library is chosen. If it is not called, the automatic library assumes that the current CPU is NOT the NS32CG16.

Each of the library functions also is usable from assembly language, simply by suffixing the emulation name from the above table with "em", setting the appropriate registers, and issuing a bsr or jsr to the routine. The assembly language interface is the same register set up as for the NS32CG16 instructions. More information on the register setup for the NS32CG16 instructions is available in the *Programmer's Reference Manual Supplement for the NS32CG16*. Below is an example of the assembly and C calling techniques:

```
        addr        source,r0       #point to the source
        addr        dest,r1                #point to the destination
        movqd        5,r2               #set up the shift amount
        cmpqb        $0,r2               #initialize the L bit
        jsr        bitwtem               #call the bitwt emulation

From C, this example would be:

        bitwt(source,dest,5); /* Call the bitwt emulation */
```

4. Argument passing

When called from C, arguments are passed by value to the emulation routines on the stack, in standard C convention.

BITBLT Group

Each of the BitBlt group of functions (*BBAND, BBFOR, BBOR, BBSTOD, BBXOR*)
is passed the same arguments, in the same order, as follows:

```
    bbxxx(source,dest,shift,height,firstmask,secondmask,swarp,dwarp,width);
```

Source is a 32-bit pointer to the source data. Dest is a 32-bit pointer to the destination data. Shift is the shift amount. Only the last 5 bits are significant, but should be in the range 0-15. Height is the unsigned 32-bit height of the BitBlt operation. Firstmask is the 16-bit first mask applied to the leading edge of the BitBlt operation. Secondmask is the 16-bit second mask applied to the trailing edge of the BitBlt operation. Swarp is the adjusted source warp (Swarp = SourceWarp - 2*(width-1)). Dwarp is the adjusted destination warp (Dwarp = DestWarp - 2*(width-1)). Width is the unsigned 16-bit quantity of the number of words of source data to operate on. Shift should be in the range 0-15. Height must be non-zero, and in the range 0 < height < 2^32. Width must be non-zero, and in the range 0 < width
< 2^16.

BITWT function

The *BITWT* emulation routine is passed only three arguments:

```
bitwt(source,dest,shift);
```

Source is the 32-bit pointer to the source data. Dest is the 32-bit pointer to the destination data. Shift is the number of bits of shift desired, and should be in the range 0 <= shift =< 15. It is strongly recommended that the *BITWT* function NOT be called from C, as the equivalent C code will operate faster in the general case. To use *BITWT* effectively, it must be unrolled into the application's most common widths and heights, and be called as a unique function. The equivalent C code is as follows:

```
unsigned short *source,*dest;
int shift;


*(unsigned long *)dest |= (*source++ << shift);
dest++;
```

EXTBLT functions

The *EXTBLT* group has two functions, one which does pre-read, and one that
does not. While this code may be used as-is, it is strongly recommended that when using the DP8510 in a non-NS32CG16 environment the circuit be

designed to use a read-write mode with an extension of the write cycle to a read-modify-write cycle, rather than a read-read-write mode. See the *EXTBLT* listings for more information.

The parameters for the *EXTBLT* functions are as shown:

```
extblt(source,dest,width,height,horzincr,swarp,dwarp);
extbltp(source,dest,width,height,horzincr,swarp,dwarp);
```

Source is a 32-bit pointer to the source data. Dest is a 32-bit pointer to the destination data. Width is the unsigned 32-bit quantity of the number of words of destination data to operate on. Height is the unsigned 32-bit height of the BitBlt operation. Horzincr is the Horizontal Increment (usually +2 or -2). Swarp is the adjusted source warp (Swarp = SourceWarp - (width in bytes -2)). Dwarp is the adjusted destination warp (Dwarp = DestWarp - (width in bytes -2)).

MOVMP functions

The *MOVMP* group allows quick filling of byte, word or double word quantities. The calling sequence is as follows:

```
movmpb(dest,pincr,count,pat);
movmpw(dest,pincr,count,pat);
movmpd(dest,pincr,count,pat);
```

Dest is a 32-bit pointer to the destination data to be affected. Pincr is the pointer increment, applied to the dest pointer after each write. Count is the unsigned 32-bit number of times to write the pattern (note: may be zero, meaning write no data). Pat is the 8, 16 or 32 bit pattern to write.

SBITPS function

The *SBITPS* emulation allows runs of bits to be set, if the distance (in bits) between each is constant. This allows horizontal, vertical, and 45° lines to be drawn. It also assists in other line drawing algorithms (see *NS32CG16 Graphics Applications Note 2, Drawing lines*). The calling

sequence
for the *SBITPS* emulation is as follows:

```
sbitps(dest,bitoff,count,dwarp);
```

Dest is a 32-bit pointer to the destination base address. Bitoff is a 32-bit signed bit offset (may be in range -2^31 < bitoff < 2^31). Count is the unsigned 32-bit count of the number of bits to be set (may be zero, meaning set no bits). Dwarp is the signed 32-bit value which is added to the bitoff following each operation. By maintaining true bit positions, this instruction may draw vertical lines (Dwarp = + or - (warp * (pixels per byte))), horizontal lines (Dwarp = + or - 1) or 45° lines (Dwarp = + or - (warp * (pixels per byte)) + or - 1).

SBITS function

The *SBITS* emulation allows a number of contiguous bits to be set in memory.
This allows horizontal lines to be created. The emulation also automatically invokes an algorithm to set large quantities of bits if the number of bits to set is above 25. This occurs in all libraries. An alternate entry point (Xsbits) is available for exact emulation of the NS32CG16 *SBITS* instruction. This entry point does nothing if the number of
bits to be set is above 25. In assembly language, this routine returns with the flags set such that the *LT* condition is set if the number of bits to be set is above 25, and the *GE* condition set if the number is less than or equal to 25. The calling sequence for *SBITS* and *Xsbits* is:

```
sbits(dest,bitoff,count,table);
Xsbits(dest,bitoff,count,table);
```

Dest is a 32-bit pointer to the destination base address. Bitoff is a 32-bit signed bit offset (may be in range -2^31 < bitoff < 2^31). Count is the unsigned 32-bit count of the number of bits to be set (may be zero, meaning set no bits). Table is the 32-bit pointer to the sbits table as described in the *Programmer's Reference Manual Supplement for the NS32CG16*. This
table is available in the library, by declaring an external array called sbitstab, as shown below:

```
extern unsigned long sbitstab[ ];
sbits(dest,200,300,sbitstab);
```

The *SBITS* instruction is significantly faster than *SBITPS* for drawing horizontal lines.

TBITS functions

As the tbits0 and tbits1 instructions return the results in the PSR, they are not useful when called from C. There is no calling sequence for these instructions, but the assembly language emulations are available.

5. Optimization notes

When using a *Series 32000* processor other than the NS32CG16 for graphics,
these emulation routines offer a good start to a fully optimized package. To continue the optimizations, here are some general hints.

Customize the instruction emulations for your particular application. If the most common character imaged, for example, is a 32 bit by 54 line character, build a `special case' version of the *BBFOR* instruction to handle this. By unrolling the loops, as has been done for the 1 and 2 wide cases in the emulation libraries, registers can be freed to hold more temporary values. If the source or destination warps are constant in a common case, code these directly into the register offsets, as is shown in the *EXTBLT* example code. To effectively reduce run time, minimize the number of instructions executed (using the two-address mode of *Series 32000*), minimize the number of memory references, and minimize the number
of loops.

6. Conclusions

The emulation routines presented in this applications note permit the user to code freely in C, and execute graphics-type instructions directly. Development and testing was performed on the SYS32/25 development system.
Other graphics routines and applications notes have been developed

successfully using these routines.

Full emulations of the NS32CG16-specific instructions allow migration of a product designed with the NS32CG16 to other *Series 32000* processors without
significant rewriting of the original code. This allows a product to have a performance span from the NS32CG16-15 through the NS32332-15 to the NS32532-30 and beyond.